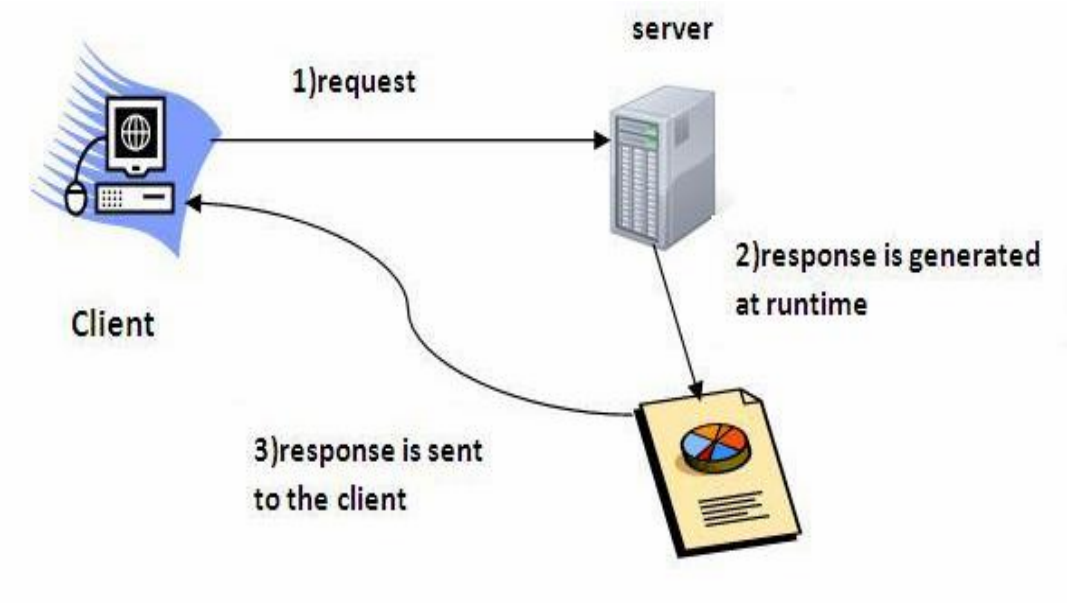
Web Technologies

UNIT – 4

Servlets

Introduction to Servlets:

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.



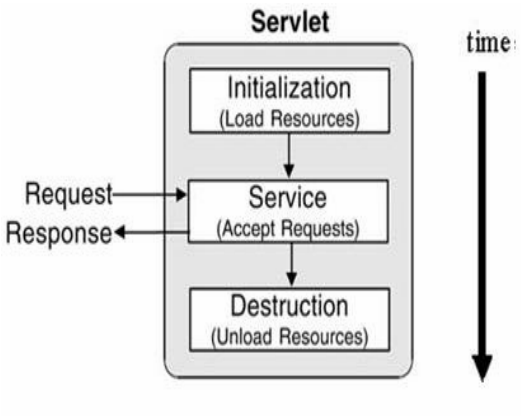
Lifecycle of a Servlet:

A servlet life cycle can be defined as the entire process from its creation till the destruction. The web container maintains the life cycle of a servlet instance. This life cycle is expressed in the API by the init, service, and destroy methods of the javax.servlet.Servlet interface that all servlets must implement directly or indirectly through the GenericServlet or HttpServlet abstract classes.

The following are the paths followed by a servlet

- The servlet is initialized by calling the **init ()** method.

- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()**method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.



init(): The web container calls the init method only once after creating the servlet instance. It is called when the servlet is first created, and not called again for each user request. The init method is used to initialize the servlet.

```
public void init(ServletConfig config) throws ServletException
{
        //code
}
```

service(): The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method.

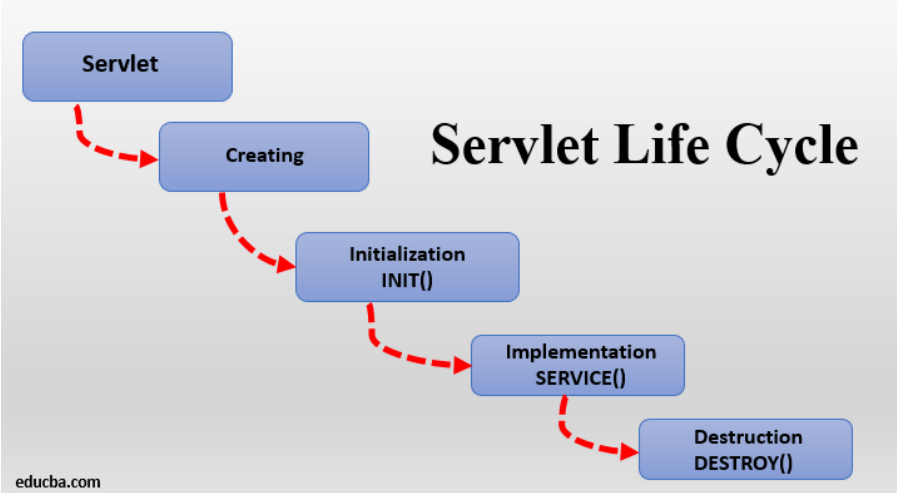
```
public void service(ServletRequest request,ServletResponse response) throwsServl
etException, IOException
{
        //code
}
```

destroy(): The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc.

```
public void destroy()
{
        //code
}
```

Servlet life cycle contains five steps:

1) Loading of Servlet
2) Creating instance of Servlet
3) Invoke init() once
4) Invoke service() repeatedly for each client request
5) Invoke destroy()



Step 1: Loading of Servlet
When the web server (e.g. Apache Tomcat) starts up, the servlet container deploy and loads all the servlets.

Step 2: Creating instance of Servlet
Once all the Servlet classes loaded, the servlet container creates instances of each servlet class. Servlet container creates only once instance per servlet class

Step 3: Invoke init() once

- The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:
- **public void** init(ServletConfig config) **throws** ServletException

Step4 : invoke service()

- The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method.

public void service(ServletRequest request, ServletResponse response)**throws** ServletExcept ion, IOException

Step 5: destroy()

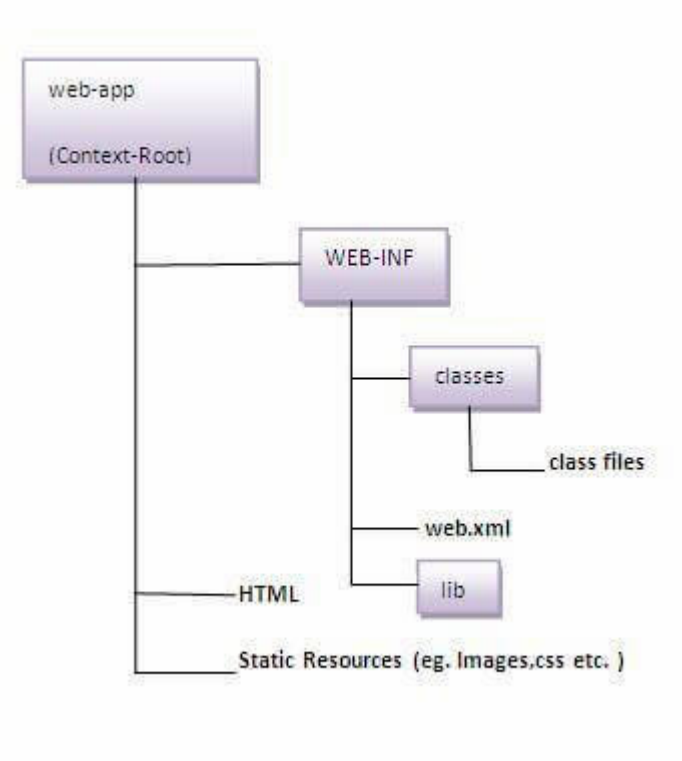
- It gives the servlet an opportunity to clean up any resource for example memory, thread etc.
- **public void** destroy()

DEVELOPING FIRST SERVLET APPLICATION

The steps are as follows:

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

CREATE A DIRECTORY STRUCTURE



CREATE A SERVLET

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");//setting the content type
PrintWriter pw=res.getWriter();//get the stream to write the data

//writing html in the stream
pw.println("<html><body>");
pw.println("Welcome to servlet");
pw.println("</body></html>");
```

```
pw.close();//closing the stream
}}
```

COMPILE THE SERVLET

CREATE DEPLOYMENT DESCRIPTOR

```
<web-app>

<servlet>
<servlet-name>FirstServlet</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>FirstServlet</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

GENERIC SERVLET CLASS

The GenericServlet class provides implementations of the basic life cycle methods for a servlet. GenericServlet implements the Servlet and ServletConfig interfaces. In addition, a method to append a string to the server log file is available. The signatures of this method are shown here:

void log(String *s*)
void log(String *s*, Throwable *e*)

Here, *s* is the string to be appended to the log, and *e* is an exception that occurred.

GenericServlet class implements Servlet, ServletConfig and Serializable interfaces. It provides the implementation of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

```
import java.io.*;

import javax.servlet.*;


public class First extends GenericServlet{

public void service(ServletRequest req,ServletResponse res)

throws IOException,ServletException{


res.setContentType("text/html");


PrintWriter out=res.getWriter();

out.print("<html><body>");

out.print("<b>hello generic servlet</b>");

out.print("</body></html>");


}

}
```

JSDK:

- Java Servlet Development Kit contains the packages that are needed to create servlets.
- A utility known as a ServletRunner is also included which enables us to test some of the servlets that are created.
- JSDK can be downloaded from sun Microsystems website at java.sun.com.

- The following are the basic steps to build and test a Servlet:
  1. Create and compile servlet source code
  2. Start the ServletRunner Utility
  3. Start a Web browser and Request the servlet.

The Servlet API:

Consists of two packages

1. **javax.servlet :** This package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.
2. **javax.servlet.http :** This package contains interfaces and classes that are responsible for http requests only.

javax.servlet package

Interface	
Servlet	Defines methods that all servlets must implement.
ServletConfig	A servlet configuration object used by a servlet container to pass information to a servlet during initialization.
ServletContext	Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.
ServletRequest	Defines an object to provide client request information to a servlet.
ServletResponse	Defines an object to assist a servlet in sending a response to the client.
SingleThreadModel	**Deprecated.** As of Java Servlet API 2.4, with no direct replacement.

Class	
GenericServlet	Defines a generic, protocol-independent servlet.
ServletInputStream	Provides an input stream for reading binary data from a client request, including an efficientreadLine method for reading data one line at a time.
ServletOutputStream	Provides an output stream for sending binary data to

	the client.
ServletException	Defines a general exception a servlet can throw when it encounters difficulty.
UnavailableException	Defines an exception that a servlet or filter throws to indicate that it is permanently or temporarily unavailable.

Servlet

Methods
• Void destroy() Called by the servlet container to indicate to a servlet that the servlet is being taken out of service. • ServletConfig getServletConfig() Returns a ServletConfig object, which contains initialization and startup parameters for this servlet.
• String getServletInfo() Returns information about the servlet, such as author, version, and copyright. • Void init(ServletConfig config) Called by the servlet container to indicate to a servlet that the servlet is being placed into service. • Void service(ServletRequest req, ServletResponse res) Called by the servlet container to allow the servlet to respond to a request.

ServletConfig

Method
• String getInitParameter(java.lang.String name) Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist. • Enumeration getInitParameterNames() Returns the names of the servlet's initialization parameters as an Enumeration of Stringobjects, or an empty Enumeration if the servlet has no initialization parameters. • ServletContext getServletContext() Returns a reference to the ServletContext in which the caller is executing. • String getServletName() Returns the name of this servlet instance.

- **javax.servlet.http package:**

Interfaces:

There are many interfaces in javax.servlet.http package. They are as follows:

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

Classes:
There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

Reading servlet paramenters /Reading Form Data:

- HTML Form uses <form> tag to work with forms.
- <form> tag uses either **GET** method or **POST** method.
- If the form uses **GET** method in the form tag, then use **doGet()** method in the servlet.
- If the form uses **POST** method in the form tag, then use **doPost()** method in the servlet.
- Instead of adding request parameters to the end of the URL, it is also possible to POSTthem as actual data.

Servlets handles form data using the following methods.

- **getParameter():** to get the value of a form parameter.
- **getParameterValues():** if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames():** Call this method if you want a complete list of all parameters in thecurrent request.

Example:

Write a servlet program to read user ‘s name from a web page and greet with user’s name.

Index.html

```
<html>

<body>

<form action="http://localhost:4040/demo/greet" method="get">

        Enter your name : <input type="text" name="uname">

        <br>    <input type="submit" value="Greet me">

</form>

</body>

</html>
```

```
importjavax.servlet.http.*;
importjavax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet
{
        public void doGet(HttpServletRequestreq,HttpServletResponse
res)throws ServletException,IOException
        {
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();
        String name=req.getParameter("uname");
        pw.println("Welcome  "+name);
        }
}
```

web.xml

```
<?xml version="1.0"?>
<web-app>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/greet</url-pattern>
</servlet-mapping>
</web-app>
```

Initialization Parameters :

- Using init parameters, It is not needed to edit the servlet file. This concept provides facility to modify web.xml file to supply init parameters.
- If the configuration information is modified from the web.xml file, we don't need to change the servlet.
- So it is easier to manage the web application if any specific content is modified from time to time.

An object of **ServletConfig** is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

Methods of ServletConfig interface

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
3. **public String getServletName():**Returns the name of the servlet.
4. **publicServletContextgetServletContext():**Returns an object of ServletContext.

To get the object of ServletConfig,use the following metod of Servlet interface.

getServletConfig()

Syntax to provide the initialization parameter for a servlet

```
<web-app>
      <servlet>
       ......

      <init-param>
            <param-name>parametername</param-name>
          <param-value>parametervalue</param-value>
      </init-param>
      ......
      </servlet>
</web-app>
```

The**<init-param>** is used to specify the initialization parameter for a servlet.

1. .java file (This is our servlet file)
2. .xml file (We mention init parameters here)

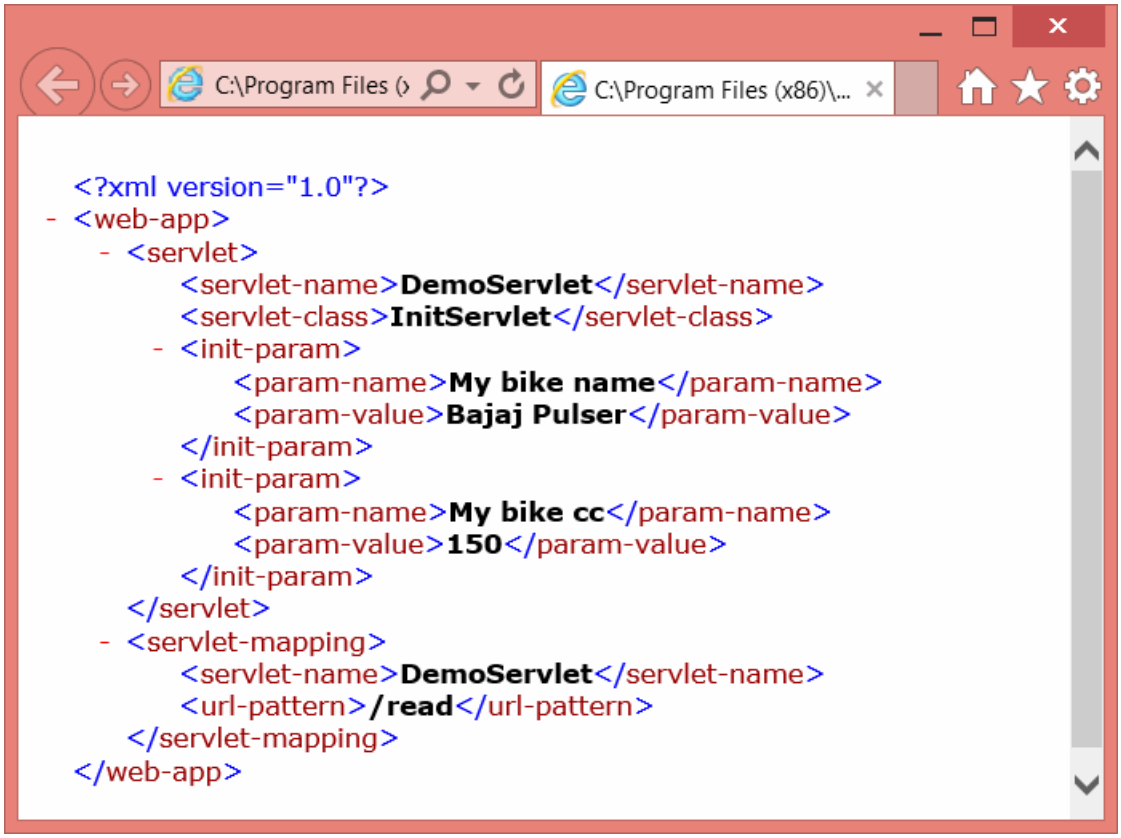
Java file must be compiled to generate a .class file.

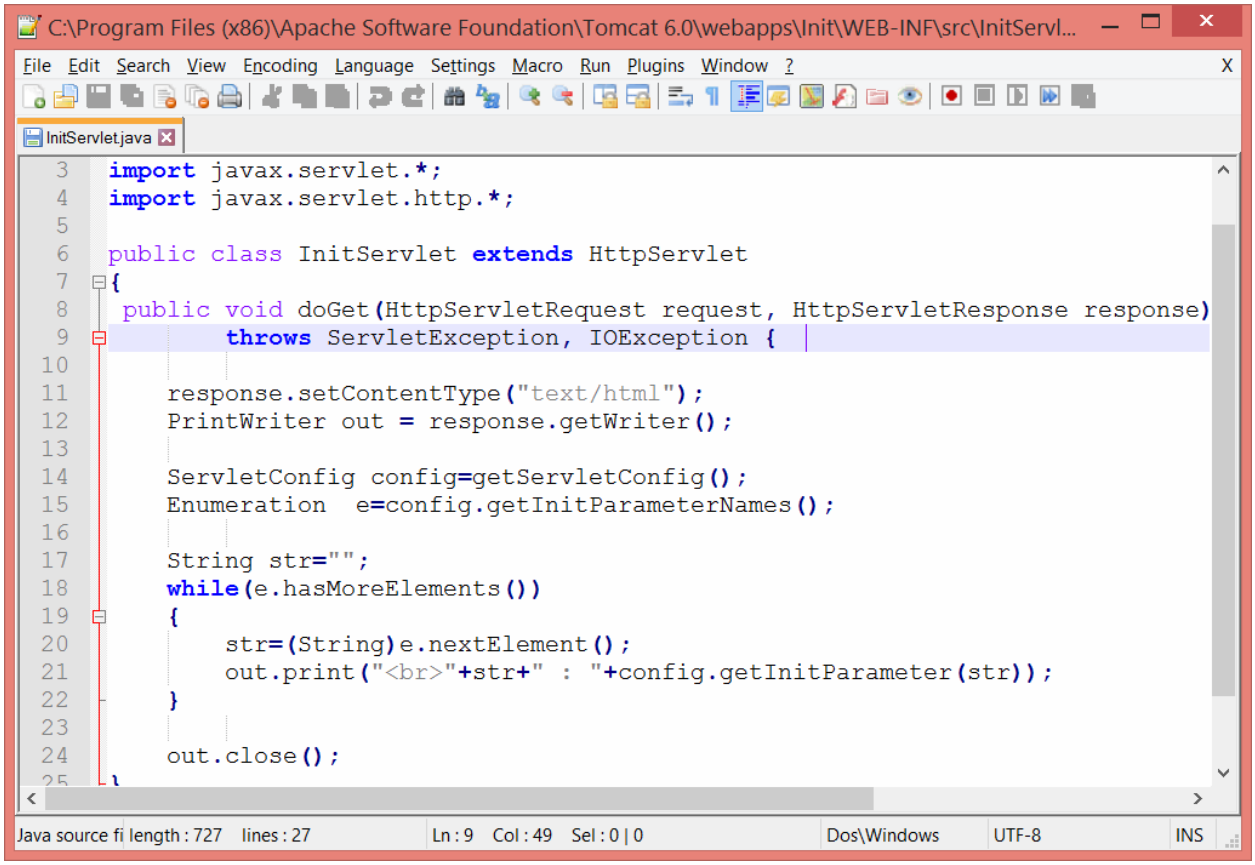
Example:

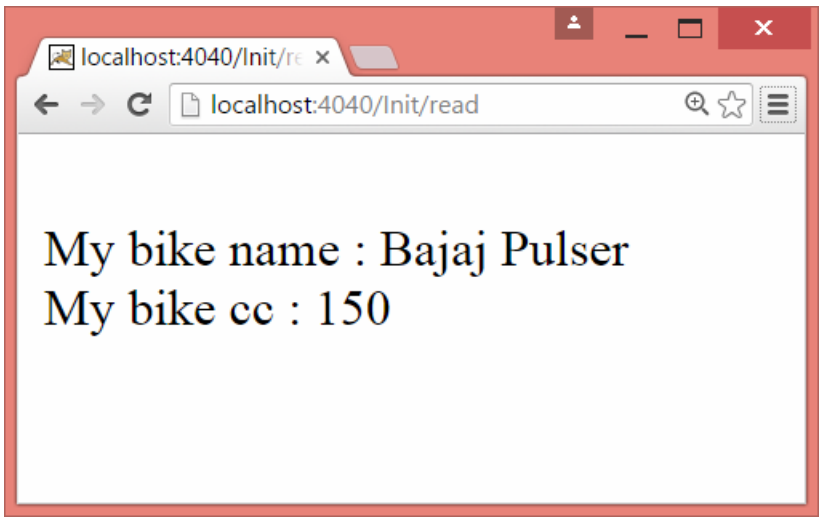
We need two files in this example.

1. InitServlet.java
2. web.xml

Compile **InitServlet.java** file to generate **InitServlet.class** and place it in classes folder.







Servlets with Database Connectivity

1. **Introduction to Servlets**
   - Java Servlets are server-side components that handle client requests and generate dynamic web responses.
   - They run on a web server (e.g., Tomcat) and extend the functionality of web applications.
2. **Steps to Connect a Servlet with a Database**

Load the JDBC Driver:

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

Establish Connection:

```
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/dbname", "user",
"password");
```

Create a Statement:

```
Statement stmt = con.createStatement();
```

Execute Queries:

```
ResultSet rs = stmt.executeQuery("SELECT * FROM users");
```

Process Results and Close Connection:

```
while(rs.next()) {
   System.out.println(rs.getString("username"));
}
con.close();
```

Using Prepared Statements (Prevent SQL Injection)

```
PreparedStatement pstmt = con.prepareStatement("SELECT * FROM users WHERE
email = ?");
pstmt.setString(1, "user@example.com");
ResultSet rs = pstmt.executeQuery();
```

Servlet Handling HTTP Requests with Database

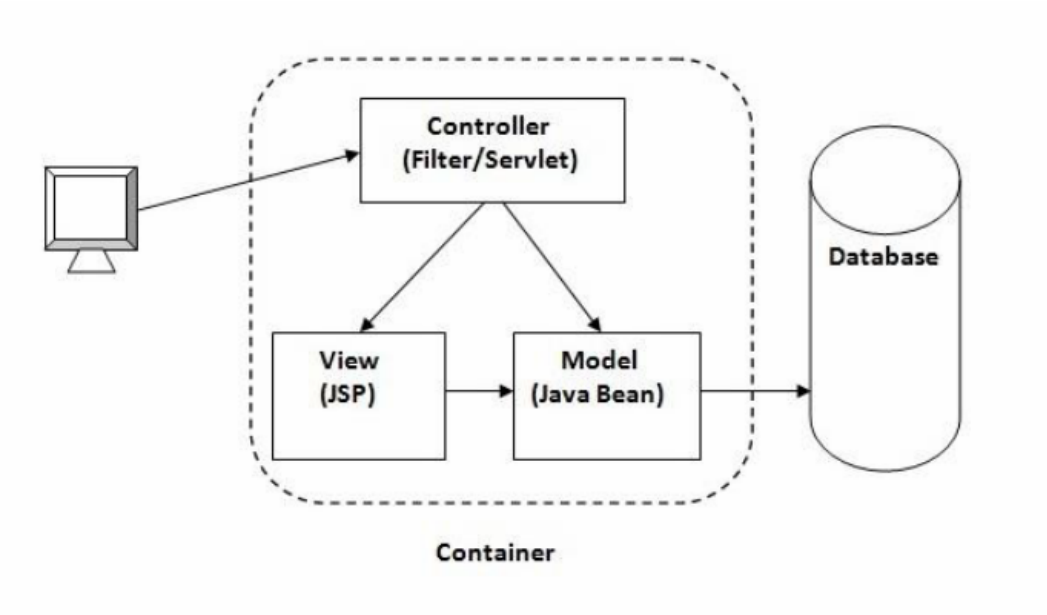
```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
   PrintWriter out = response.getWriter();
   try {
      Class.forName("com.mysql.cj.jdbc.Driver");
      Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/dbname", "user",
"password");
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT * FROM users");
      while(rs.next()) {
         out.println(rs.getString("username"));
      }
      con.close();
   } catch(Exception e) {
      out.println("Error: " + e);
```

```
    }
}
```

Introduction to Model View Controller (MVC) Architecture

1. Introduction to MVC Architecture

- **MVC (Model-View-Controller)** is a **design pattern** used in software engineering to separate concerns in an application.
- It provides a **structured approach** to application development, improving maintainability, reusability, and scalability.
- **Commonly used in web applications** (e.g., Java, .NET, PHP frameworks).



2. Components of MVC Architecture

1. **Model (Data & Business Logic)**
    - Represents the **data** and the **business logic** of the application.
    - Interacts with the database to **retrieve, update, and delete data**.
    - Ensures data consistency and applies business rules.
    - **Example**: A Java class that interacts with a database using JDBC.
2. **View (User Interface)**
    - Responsible for **displaying the data** to the user.
    - Uses technologies like **HTML, JSP, CSS, JavaScript** for UI rendering.
    - Does not contain business logic; it only represents the output.
    - **Example**: A JSP page that displays user information.
3. **Controller (Request Handling & Flow Control)**
    - Acts as an **intermediary between Model and View**.
    - **Handles user requests**, processes input, and updates the Model.
    - Decides which View to render based on the response from the Model.
    - **Example**: A Java Servlet that processes a login request and redirects to the appropriate JSP page.

3. Working of MVC Architecture

1. **User sends a request** through the View (e.g., submits a login form).
2. **Controller receives the request** and calls the appropriate Model method.
3. **Model interacts with the database** to fetch/process data.
4. **Model sends the response back to the Controller**.
5. **Controller forwards the data to the View** for display.
6. **View presents the data** to the user in a readable format.

1. **What is MVC?**
   - MVC is a design pattern used in web applications for separating concerns:
     - **Model**: Handles data and business logic (e.g., database interactions).
     - **View**: Manages UI representation (HTML, JSP, etc.).
     - **Controller**: Processes user requests and interacts with Model and View.
2. **Benefits of MVC**
   - **Separation of Concerns**: Easier maintenance and testing.
   - **Reusability**: Components can be reused independently.
   - **Scalability**: More organized and scalable web applications.
3. **MVC Flow in Java Web Applications**
   - **User requests a URL** → Sent to a **Servlet (Controller)**
   - **Servlet interacts with the Model (Database or Business Logic Layer)**
   - **Model processes data and sends it back to the Controller**
   - **Controller forwards data to a JSP (View) using RequestDispatcher**
   - **JSP renders the response to the user**
4. **Example of MVC Implementation**

   Controller (Servlet)

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
   String username = request.getParameter("username");
   UserModel user = UserDAO.getUser(username);  // Calls Model
   request.setAttribute("user", user);
   RequestDispatcher rd = request.getRequestDispatcher("user.jsp"); // Forwards
to View
   rd.forward(request, response);
}
```

   Model (DAO Layer)

```
public class UserDAO {
   public static UserModel getUser(String username) {
      try {
         Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/db", "user", "pass");
```

```
        PreparedStatement ps = con.prepareStatement("SELECT * FROM users
WHERE username=?");
        ps.setString(1, username);
        ResultSet rs = ps.executeQuery();
        if(rs.next()) {
            return new UserModel(rs.getString("username"), rs.getString("email"));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
  }
}
```

View (JSP Page: user.jsp)

```
<% UserModel user = (UserModel) request.getAttribute("user"); %>
<h1>Welcome, <%= user.getUsername() %></h1>
<p>Email: <%= user.getEmail() %></p>
```